Forbidden Configurations: Finding the number predicted by the Anstee-Sali Conjecture is NP-hard

Miguel Raggi [†]

Centro de Ciencias Matemáticas
Universidad Nacional Autónoma de México,
Morelia, Michoacán, México

Let F be a hypergraph and let $\mathrm{forb}(m, F)$ denote the maximum number of edges a hypergraph with m vertices can have if it doesn't contain F as a subhypergraph. A conjecture of Anstee and Sali predicts the asymptotic behaviour of $\mathrm{forb}(m, F)$ for fixed F. In this paper we prove that even finding this predicted asymptotic behaviour is an NP-hard problem, meaning that if the Anstee-Sali conjecture were true, finding the asymptotics of $\mathrm{forb}(m, F)$ would be NP-hard.

Mathematics Subject Classification: 05D05

Keywords: Forbidden Configuration, Hypergraph, Trace, NP-hard, NP-complete, Anstee-Sali conjecture

1 Introduction

The paper considers an extremal problem. Some of the most celebrated extremal results are those of Erdős and Stone [ES46] and Erdős and Simonovits [ES66]. They consider the following problem: Given $m \in \mathbb{N}$ and a graph F, find the maximum number of edges in a graph on m vertices that avoids having a subgraph isomorphic to F.

There are a number of ways to generalize this to hypergraphs. A k-uniform hypergraph is one in which each edge has size k. Some view k-uniform hypergraphs as the most natural generalization of a graph (a graph is a 2-uniform hypergraph) and one might also generalize the forbidden subgraph problem to a forbidden k-uniform subhypergraph problem. There are both asymptotic results and exact bounds (e.g. [dCF00], [Pik08], [Für91]).

Forbidden Configurations is a different (but also natural) generalization that is studied mainly by Richard Anstee and his colaborators. We consider the following problem: Given $m \in \mathbb{N}$ and a hypergraph F, find the maximum number of edges in a simple hypergraph H on m vertices that avoids having a subhypergraph isomorphic to F.

[†]Email: `mraggi@gmail.com`. Research supported by DGAPA Postdoctoral scholarship, UNAM

We find it convenient to use the language of matrices to describe hypergraphs: Each column of a $\{0,1\}$-matrix can be viewed as an incidence vector on the set of rows.

Definition 1.1. Define a matrix to be **simple** if it is a $\{0,1\}$-matrix with no repeated columns. Then an $m \times n$ simple matrix corresponds to a **simple hypergraph** (or **set system**) on m vertices with n distinct edges where we allow the empty edge. Let $\|A\|$ denote the number of columns in A (which is the cardinality of the associated set system).

Definition 1.2. Let A and B be $\{0,1\}$-matrices with the same number of rows. Define the **concatenation** $[A|B]$ to be the configuration that results from taking all columns of A together with all columns of B. For $t \in \mathbb{N}$, we define the product

$$t \cdot A := [\underbrace{A \mid A \mid \cdots \mid A}_{t \text{ times}}].$$

Our objects of study are $\{0,1\}$-matrices with row and column order information stripped from them:

Definition 1.3. Two $\{0,1\}$-matrices are said to be **equivalent** if one is a row and column permutation of the other. This defines an equivalence relation. An equivalence class is called a **configuration**.

Abusing notation, we will commonly use matrices (representatives) and their corresponding configurations interchangeably.

Definition 1.4. For a configuration F and a $\{0,1\}$-matrix A (or a configuration A), we say that F is a **subconfiguration** of A, and write $F \prec A$ if there is a representative of F which is a submatrix of A. We say A **has no configuration** F (or **doesn't contain** F **as a configuration**) if F is not a subconfiguration of A. Let $\mathrm{Avoid}(m,F)$ denote the set of all m-rowed simple matrices with no subconfiguration F.

Our main extremal problem is to compute

$$\mathrm{forb}(m,F) = \max_A\{\|A\| \ : \ A \in \mathrm{Avoid}(m,F)\}.$$

Perhaps some examples are useful:

- $\mathrm{forb}(m, \begin{bmatrix}1\\1\end{bmatrix}) = m+1$, since we can take all columns with at most one 1.
- $\mathrm{forb}(m, \begin{bmatrix}1\\0\end{bmatrix}) = 2$, since we may only take the column of 1's and the column of 0's (*i.e.* the empty set and the complete set).
- $\mathrm{forb}(m, \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}) = \binom{m}{2} + \binom{m}{1} + \binom{m}{0}$, by taking all columns with at most two 1's. The proof that this is indeed the maximum is easy and can be found in [Rag11].

Surveys on the topic can be found in [Ans] and [Rag11]. Let A^c denote the $\{0,1\}$-complement of A (replace every 0 in A by a 1 and every 1 by a 0). Note that $\mathrm{forb}(m,F) = \mathrm{forb}(m,F^c)$.

Remark 1.5. Let F and G be configurations such that $F \prec G$. Then $\mathrm{forb}(m,F) \le \mathrm{forb}(m,G)$.

We say a column α has **column sum** t if it has exactly t ones. Define $\mathbf{0}_m$ to be a column with m 0's and $\mathbf{1}_m$ to be a column of m 1's.

For a set of rows S, we let $A|_S$ denote the submatrix of A given by restricting the rows of A to only those in S.

An important general result due to Füredi applies to simple or to non-simple configurations.

Theorem 1.6 (Z. Füredi). Let F be a given k-rowed $\{0,1\}-$matrix. Then $\mathrm{forb}(m,F)$ is $O(m^k)$.

We desire more accurate asymptotic bounds. Anstee and Sali conjectured that the best asymptotic bounds can be achieved with certain *product constructions*.

Definition 1.7. Let A and B be $\{0,1\}$-matrices. We define the **product** $A \times B$ by taking each column of A and putting it on top of every column of B. Here is an example of a product:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \implies \begin{matrix} A \\ \times \\ B \end{matrix} = \left[\begin{array}{cc|cc|cc} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right].$$

Note that this is a well defined operation in configurations.

We are interested in asymptotic bounds for $\mathrm{forb}(m,F)$. Let $\mathcal{I}_m$ be the $m \times m$ identity matrix, $\mathcal{I}_m^c$ be the $\{0,1\}$-complement of $\mathcal{I}_m$ (all ones except for the diagonal) and let $\mathcal{T}_m$ be the tower matrix: a matrix corresponding to a maximum chain in the partially ordered set of the power set of the vertices. For example,

$$\mathcal{T}_4 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Anstee and Sali conjectured that the asymptotically "best" constructions avoiding a single configuration would be products of $\mathcal{I}, \mathcal{I}^c$ and $\mathcal{T}$.

Conjecture 1.8. [AS05] Let F be a configuration. Let

$$P_r(a,b,c) := \underbrace{\mathcal{I}_r \times \ldots \times \mathcal{I}_r}_{a \text{ times}} \times \underbrace{\mathcal{I}_r^c \times \ldots \times \mathcal{I}_r^c}_{b \text{ times}} \times \underbrace{\mathcal{T}_r \times \ldots \times \mathcal{T}_r}_{c \text{ times}},$$

Define $X(F)$ to be the largest number such that there exist numbers $a,b,c \in \mathbb{N}$ with $a+b+c = X(F)$ such that for all $r \in \mathbb{N}$,

$$F \not\prec P_r(a,b,c).$$

Then $\mathrm{forb}(m,F)$ is $\Theta(m^{X(F)})$.

Observe that $X(F)$ is always an integer. Also note that $\|P_r(a,b,c)\| = r^{a+b} \cdot (r+1)^c$ which is $\Theta(r^{X(F)})$, so by taking $r = \lceil m/X(F) \rceil$ (and perhaps deleting some rows in case $X(F) \nmid m$), we have that $\|P_r(a,b,c)\|$ is $\Omega(m^{X(F)})$, so the fact that $\mathrm{forb}(m,F)$ is $\Omega(m^{X(F)})$ is built into the conjecture. In order to prove the conjecture, all that would be required would be to prove that $\mathrm{forb}(m,F)$ is $O(m^{X(F)})$ for every F. A disproof could be potentially easier, as only a counterexample would be required.

The conjecture has been proven for all $k \times \ell$ configurations F with $k \in \{1, 2, 3\}$ and many others cases in various papers. The proofs for $k = 2$ are in [AGS97], for $k = 3$ in [AGS97], [AFS01], [AS05]. For $\ell = 2$, the conjecture was verified in [AK06]. For $k = 4$, all cases either when the conjecture predicts a cubic bound for F or when F is simple were completed in [AF10]. For $k = 4$ and F non-simple, there are three boundary cases with quadratic bounds, one of which is established in [ARS12]. For $k \in \{5, 6\}$ some results can be found in [ARS11].

Anstee has long conjectured that even finding $X(F)$ given F was not a trivial task, and the question of its NP-hardness was long conjectured but not proven ([Ans], [Rag11]). In this paper we settle that finding X is indeed NP-hard and we also note that one of the decision versions associated with this optimization problem is NP-complete, adding this function to the long list of functions known to be NP-complete, with the interesting plus that this function is conjectured to give the *exponent* of the asymptotic growth of forb.

For relatively small configurations F we have a computer program that yields the answer (relatively) quickly. The source code (in C++) can be freely downloaded from:

`http://matmor.unam.mx/~mraggi/`

This program can compute $X(F)$ for F having less than $\sim$10 rows in just a few minutes. This task takes merely exponential time, not doubly exponential (as it is often the case with forbidden configuration problems). This program was written to perform many other tasks other than finding $X(F)$. A description of the algorithm used for this task is in the appendix 4.

2 Results

There are two natural decision problems associated with $X(F)$: Given F and k as inputs,

1. Is it true that $X(F) < k$?
2. Is it true that $X(F) \geq k$?

We prove that the first of the two decision problems is in NP by exhibiting a certificate which can be checked in polynomial time.

The main result of this paper is the following:

Theorem 2.1. Finding $X(F)$ is an NP-hard problem. In other words, should a polynomial-time algorithm exist for finding $X(F)$ given F, then every problem in NP could be solved in polynomial time. Furthermore, the problem of "given F and k, is $X(F) < k$?" is in NP.

Before proving this theorem, we need a few lemmas.

Observation 2.2. Let F be a configuration with n rows. Then $X(F) \leq n$. Indeed, assume for the sake of contradiction that a, b, and c are such that $a + b + c = n + 1$ and $F \not\prec P_r(a, b, c)$. We may place each row of F each into a different factor of the product. The extra matrix ensures the repeated columns of F get repeated as many times as needed.

This observation in particular implies that if a polynomial time algorithm existed for any of the two decision versions of the problem, then we'd have a polynomial time algorithm for finding $X(F)$, which together with Theorem 2.1 would make the decision version of finding $X(F)$ an NP-complete problem.

A simple (but surprising) corollary of conjecture 1.8, if it were true, would be that repeating columns more than twice in F has no effect on the asymptotic behavior of $\mathrm{forb}(m, F)$. In other words, assuming the conjecture were true, the multiplicity of a column in a configuration would not affect the asymptotic bound and, asymptotically, it would only matter if a column is not there (has multiplicity 0), appears once (has multiplicity 1), or appears “multiple times” (has multiplicity 2 or more). Formally,

Proposition 2.3. Let $F_t = [G|t \cdot H]$ with G and H simple $\{0,1\}$-matrices that have no columns in common. Then $X(F_2) = X(F_t)$ for all $t \geq 2$. In particular, if the conjecture were true, then $\mathrm{forb}(m, F_t)$ and $\mathrm{forb}(m, F_2)$ would have the same asymptotic behavior.

Proof: It suffices to show that given t, G, H, a, b and c, there exists an R such that for every $r \geq R$, we have

$$F_2 = [G|2 \cdot H] \prec P_r(a,b,c) \quad \iff \quad F_t = [G|t \cdot H] \prec P_r(a,b,c).$$

Since $F_2 \prec F_t$, we only need to prove that if $F_2 \prec P_r(a,b,c)$ for some r, then $F_t \prec P_R(a,b,c)$ for some R. Suppose then F_2 is contained in the product $P_r(a,b,c)$ for some r. The idea is to find a subconfiguration of $P_r(a,b,c)$ in which there are some columns with multiplicity 1, and for the columns with multiplicity 2 or more, the multiplicity depends on r. We need r large enough so that the multiplicity of any one column (with multiplicity of 2 or more) is larger than t. Let x be the number of rows of F_t. Notice the following three facts, which include definitions for $E_{\mathcal{I}}$, $E_{\mathcal{I}^c}$ and $E_{\mathcal{T}}$.

$$\begin{aligned} E_{\mathcal{I}}(x,r) &:= [(r-x)\cdot \mathbf{0}_x \mid \mathcal{I}_x] \prec \mathcal{I}_r \\ E_{\mathcal{I}^c}(x,r) &:= [(r-x)\cdot \mathbf{1}_x \mid \mathcal{I}^c_x] \prec \mathcal{I}^c_r \\ E_{\mathcal{T}}(x,r) &:= \left\lfloor \frac{r}{x} \right\rfloor \cdot \mathcal{T}_x \prec \mathcal{T}_r. \end{aligned}$$

The first and second facts are easy to see; just take any subset of x rows from $\mathcal{I}_r$ or $\mathcal{I}^c_r$. The third statement is true by taking the $\lfloor r/x \rfloor$-th row of $\mathcal{T}_r$, the $2\lfloor r/x \rfloor$-th row of $\mathcal{T}_r$, etcetera, up to the $x\lfloor r/x \rfloor$-th row. For example, if $r = 5$ and $x = 2$, we may take the second and fourth row from $\mathcal{T}_5$:

$$\mathcal{T}_5 = \begin{bmatrix} 0&1&1&1&1&1\\ 0&0&1&1&1&1\\ 0&0&0&1&1&1\\ 0&0&0&0&1&1\\ 0&0&0&0&0&1 \end{bmatrix} \quad \Longrightarrow \quad \mathcal{T}_5|_{\{2,4\}} = \begin{bmatrix} 0&0&1&1&1&1\\ 0&0&0&0&1&1 \end{bmatrix} = E_{\mathcal{T}}(2,5)$$

Note that in the three configurations $E_{\mathcal{I}}(x,r), E_{\mathcal{I}^c}(x,r)$ and $E_{\mathcal{T}}(x,r)$, we have that there are some columns of multiplicity 1 and there are some columns for which their multiplicity can be made as large as we wish by making r large. Formally, let $E(x,r)$ be one of $E_{\mathcal{I}}(x,r)$ or $E_{\mathcal{I}^c}(x,r)$ or $E_{\mathcal{T}}(x,r)$. We have that for every x-rowed column α there are three possibilities: either $\lambda(\alpha, E(x,r)) = 0$ for all r, or $\lambda(\alpha, E(x,r)) = 1$ for all r, or $\lim\limits_{r\to\infty} \lambda(\alpha, E(x,r)) = \infty$.

If α is a column for which $\lim\limits_{r\to\infty} \lambda(\alpha, E(x,r)) = \infty$, we may conclude that there is an R for which $\lambda(\alpha, E(x,r)) \geq t$ for every $r \geq R$.

Since F_2 is contained in $P_r(a,b,c)$ for some r, the columns in H will have multiplicity at least 2 in some subset of the rows of $P_r(a,b,c)$. We see that F_t is also a subconfiguration of $P_R(a,b,c)$. □

3 Proof of the Main Theorem

We now prove the main theorem.

Proof: First we prove that the decision problem has a certificate which can be checked in polynomial time. A certificate that indeed $X(F) < k$ would have to be a proof that $F \prec P_r(a,b,c)$ for each triple a,b,c for which $a+b+c=k$. Note that there are at most a quadratic (with respect to the number of rows) number of a,b,c's which satisfy the equation, since the question has a trivial "yes" answer when k more than the number of rows (Observation 2.2).

Given F and A configurations, one can easily construct a certificate that a configuration F is indeed a subconfiguration of a configuration A: explicitly state which permutation of F appears in exactly which rows and columns of A. For the case $A = P_r(a,b,c)$, a certificate only needs to specify which rows of F go inside which factors, so at most a quadratic number of these certificates-of-being-a-subconfiguration suffice.

We now prove that finding $X(F)$ is NP-hard. Suppose there existed some polynomial-time algorithm that finds $X(F)$ given F. We shall prove that there would then exist a polynomial time algorithm for GRAPH COLORING. Suppose we are given a graph G and we wish to find the minimum number of colors for which there exists a good coloring of the graph. We may assume no isolated vertices.

The idea is to construct a 3-part matrix $F(G)$ in which the first two parts ensure there is no $\mathcal{T}$ or $\mathcal{I}^c$ in a maximum product of the form $P_r(a,b,c)$ with no subconfiguration $F(G)$, and the last part is constructed so that a partition into $\mathcal{I}'s$ produces a partition of the vertices of the graph into independent sets and vice-versa.

Suppose G has n vertices and e edges. Let M be a large number with $M \geq n+2$ and let S be the incidence matrix of G (*i.e.*, the edges of G are encoded as columns with two 1's corresponding to the vertices that belong to the edge). Construct the following simple matrix:

$$F(G) := \left[\begin{array}{c|c} 1 & \mathbf{0}_M \mathcal{I}_M^c \\ \hline 1 & \mathcal{T}_M \\ \hline S & 0 \end{array}\right]$$

Clearly we can construct $F(G)$ in polynomial time (with respect to the number of vertices of G). We prove now that we have $\chi(G) = X(F) - 2M + 1$, which in turn would yield a polynomial time algorithm for GRAPH COLORING, provided we had a polynomial time algorithm for $X(F)$.

Now, let us study the possibilities for a product of type $P_r(a,b,c)$ that does not have $F(G)$ as a subconfiguration for any r. If $b \neq 0$, then we could place all of $[1|\mathcal{I}_M^c]$ in the $\mathcal{I}^c$ part of $P_r(a,b,c)$, so $a+b+c$ would be at most $1+M+n$ (using Lemma 2.2). The same is true when $c \neq 0$. But if we let $b=c=0$, $P_r(a,0,0)$ is just a product of $\mathcal{I}$'s, so let us calculate how many $\mathcal{I}'s$ we can multiply together and still not create a subconfiguration $F(G)$.

In order for $F(G)$ to be a part of a product of $\mathcal{I}'s$, every row of $[1|\mathcal{I}_M^c]$ and $[1|\mathcal{T}_M]$ must be in a separate factor $\mathcal{I}$, since there is no $\begin{bmatrix}1\\1\end{bmatrix}$ in $\mathcal{I}$ (and also separate from the rows of $[S|0]$, since we are assuming G has no isolated vertices).

Then two rows of the $[S|0]$ part can be in the same $\mathcal{I}$ if and only if there is no $\begin{bmatrix}1\\1\end{bmatrix}$ in those two rows, which, in terms of the graph, means there is no edge between those two vertices. In other words, partitioning $[S|0]$ into I's is equivalent to partitioning the vertices of G into independent sets. So if the graph G cannot be colored with $\chi(G)-1$ colors and this is the maximum, this means that $X(F)=a=2M+\chi(G)-1\geq n+M+1$. Then $\chi(G)=X(F(G))-2M+1$. $\square$

References

[AF10] R.P. Anstee and B. Fleming, *Two refinements of the bound of Sauer, Perles and Shelah and Vapnik and Chervonenkis*, Discrete Mathematics **310** (2010), 3318–3323.

[AFS01] R.P. Anstee, R. Ferguson, and A. Sali, *Small forbidden configurations II*, Electronic Journal of Combinatorics **8** (2001), R4 25pp.

[AGS97] R.P. Anstee, J.R. Griggs, and A. Sali, *Small forbidden configurations*, Graphs and Combinatorics **13** (1997), 97–118.

[AK06] R.P. Anstee and P. Keevash, *Pairwise intersections and forbidden configurations*, European Journal of Combinatorics **27** (2006), 1235–1248.

[Ans] R.P. Anstee, *A survey of forbidden configuration results*, http://www.math.ubc.ca/∼anstee/.

[ARS11] R.P. Anstee, Miguel Raggi, and Attila Sali, *Forbidden configurations: Quadratic bounds*, preprint. (2011).

[ARS12] R.P. Anstee, M. Raggi, and A. Sali, *Evidence for a forbidden configuration conjecture: One more case solved*, Discrete Mathematics **312** (2012), no. 17, 2720 – 2729.

[AS05] R.P. Anstee and A. Sali, *Small forbidden configurations IV*, Combinatorica **25** (2005), 503–518.

[dCF00] Dominique de Caen and Z. Füredi, *The maximum size of 3-uniform hypergraphs not containing a Fano plane*, Journal of Combinatorial Theory, Series B **78** (2000), 274–276.

[ES46] P. Erdős and A.H. Stone, *On the structure of linear graphs*, Bulletin of the American Mathematical Society **52** (1946), 1089–1091.

[ES66] Paul Erdős and Miklós Simonovits, *A limit theorem in graph theory.*, Studia Scientiarum Mathematicarum Hungarica **1** (1966), 51–57.

[Für91] Z. Füredi, *Turán type problems*, Surveys in Combinatorics (Proc. of the 13th British Combinatorial Conference), ed. A.D. Keedwell, Cambridge Univ. Press. London Math. Soc. Lecture Note Series **166** (1991), 253–300.

[Pik08] O. Pikhurko, *An exact bound for Turán result for the generalized triangle*, Combinatorica **28** (2008), 187–208.

[Rag11] Miguel Raggi, *Forbidden configurations*, Ph.D. thesis, University of British Columbia, 2011.

4 Appendix: Algorithm to find X(F)

In this section we describe the algorithm used by the software described in the introduction. It runs in exponential time, of course, but it has been helpful for finding *boundary configurations* with given asymptotic bounds (see [ARS11], [Rag11] and [Ans]). Perhaps this program might be used to find a counter example to the Anstee-Sali conjecture, provided one exists, and it isn't very large.

4.1 Representation of a Configuration

We are interested in an efficient representation for configurations in order to perform the tasks described above. In the progress of our investigations, we have had various versions of the program.

Most of what we want the program to do involves performing a huge number of *configuration comparison operations*, which is testing whether or not a configuration F is a subconfiguration of a configuration A. As a first approach it would seem as if, for this task, we would be required to test each row and column permutation of F against each submatrix of A. This is of course a very slow way to do this. A simple trick to speed up the computations is to keep the columns of a configuration always in some canonical order. Then, to test whether or not a configuration F is contained in another A, we just need to permute rows of F and take subsets S of rows of A and place the columns of $A|_S$ in canonical order.

Most of the tasks we wish this program to perform involve checking whether or not a given (fixed) configuration F is a subconfiguration of a vast number of configurations A. In particular, any pre-processing we do on F can be considered as almost free. For example, finding all row permutations of F and storing them would need to be done once for each configuration F, and not at all for configurations A.

After many attempts and experiments, it seemed that the best (fastest) way to store a configuration that made many of the other tasks reasonably fast is this: Maintain an array of integers where the *indices* of the array, written in binary, are the *columns* of the configuration, and the actual numbers of the array represent the number of times a column appears. That is to say, a configuration F in m rows is represented by an array (C++ vector) $\mathbb{F}$ of size 2^m. For a number α, consider the binary representation of α and consider it as a column with m rows. If necessary, put enough 0's at the beginning of the binary representation in order to have the required m bits. The number $\mathbb{F}[\alpha]$ (the α-th number of the array) represents the number of times that column α appears in configuration F. In the implementation, we use an array of *unsigned characters* instead of integers, since we never need a configuration with the same column repeated more than 255 times. An unsigned character consists of 1 byte (8 bits).

For example, the array $\mathbb{F} = [1, 0, 0, 2, 0, 1, 0, 1]$ represents the following configuration (notice it has 3 rows, since the array has size $8 = 2^3$):

$$F = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

To see this, remember we start from 0. There is a one in position $0 = 000_b$, meaning the colum $(0, 0, 0)^T$ gets repeated one time. A two in position $3 = 011_b$, a one in position $5 = 101_b$ and a one in position $7 = 111_b$. The columns of this matrix are the representations of these numbers in binary form.

An observant reader might complain that this has the disadvantage that it requires storing 2^m bytes, and if F doesn't have many columns, most of those will be 0's. But it's a minor disadvantage, because even at 10 rows we would only need 1024 bytes, and we usually have configurations for which the number of rows is 5 or less (32 bytes). Perhaps this would become more of an issue with configurations with a high number of rows, but for those configurations, most of our tasks would require too much time to be of any practical use anyway.

We came to this representation after an implementation which represented columns as an array of bits (C++ bitset) and storing them into an ordered tree-like structure (C++ multiset from the STL). This might be a more natural implementation, but profiling the code made clear that the program was spending most of its time counting how many columns of a certain type appeared in a configuration, and was also spending a considerable amount of time navigating the tree. Explicitly storing the number of times each column appears, and making that number instantly accessible by storing it in an array (for random access) gives a very noticeable speedup and allows us to consider larger problems. By representing columns as numbers, we can do a lot of preprocessing and compute large tables in which we have almost instant access time.

For example, consider the following problem, which has to be done many times for our tasks: Given a column α and a subset S (represented by an integer as written in binary), what column is α_S? This is relatively slow to compute, but we can fill out a table by preprocessing to speed up any further access to it. Since we do this a few million times, the investment is sound.

The other advantage is that it becomes immediately clear how to compare two configurations with the same number of rows to see if one is a column-permutation submatrix of the other; check if for any column (index) the integer at position c of the first array is bigger than that of the second array. To check if F is a subconfiguration of A, we would need to find all permutations of the rows of F (which we need to do just once per configuration).

4.2 Subconfigurations

Suppose F and A are configurations and we want to decide if $F \prec A$. Then for every s-tuple of rows S (where s is the number of rows of F), we can extract from A the configuration $A|_S$ easily with our pre-stored table of columns and subsets. Once we've done this for each column of A and found $A|_S$, then for each permutation of rows of F, we check if every column α in the array corresponding to configuration F appears less than or equal to the corresponding number for column α in the array of $A|_S$. We can check every subset S like this. If at any point this is so, we can return true.

There are a few speedups. Sometimes it's immediately obvious a configuration can't be contained in another. For example, if there are more 1's in F than in A, or if F has more columns or rows than A, then $F \not\prec A$.

4.3 Determining X(F)

Given a configuration F, we wish to find $X(F)$. In other words, we wish to find the conjectured asymptotic bound for $\mathrm{forb}(m, F)$.

We may make a simplification using Proposition 2.3 and assume the multiplicity of any column of F is at most 2.

First, suppose we wanted to test whether or not there exists r such that configuration F is contained in the product

$$P_r(a,b,c) = \underbrace{I_r \times \ldots \times I_r}_{a \text{ times}} \times \underbrace{I_r^c \times \ldots \times I_r^c}_{b \text{ times}} \times \underbrace{T_r \times \ldots \times T_r}_{c \text{ times}}.$$

Building this object with $r = R$ as calculated in Proposition 2.3 would be prohibitively slow. Instead, we build a set $\mathcal{X}$ of subconfigurations from $P_R(a,b,c)$ such that if $F \prec P_R(a,b,c)$, then $F \prec X$ for some $X \in \mathcal{X}$.

Recall that if $F \prec P_R(a,b,c)$, then the rows of F get partitioned into $a+b+c$ parts (a part can be empty), where each part belongs to a factor of the product $P_R(a,b,c)$. Because of Proposition 2.3, we can assume each column appears at most twice. Given $s \in \mathbb{N}$, consider the following matrices:

$$A_I(s) = \begin{bmatrix} t \cdot 0_s & I_s \end{bmatrix} \ , \quad A_{I^c}(s) = \begin{bmatrix} t \cdot 1_s & I_s^c \end{bmatrix} \ , \quad A_T(s) = t \cdot \begin{bmatrix} T_s \end{bmatrix} .$$

We see that an s-rowed configuration F (with each column repeated at most t times) is contained in I_m for some large m, if and only if $F \prec A_I(s)$. We can then consider all partitions of rows of F and see if each part is contained in the corresponding A_I, A_{I^c} or A_T.

For example, to test whether

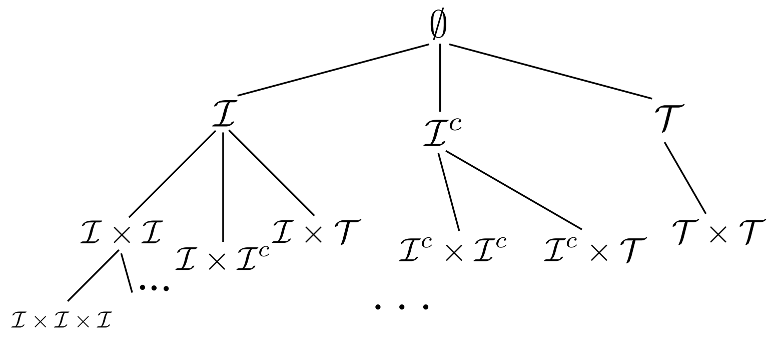
$$F = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

is contained in $I \times T \times T$, we would partition the rows of F in three parts. In this case, F has five rows, so consider, for example, the following partition of 5: $(2,2,1)$. Consider the following representatives:

$$A_I(2) = \left[s \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} \ \middle| \ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right] \ , \quad A_T(2) = s \cdot \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_T(1) = s \cdot \begin{bmatrix} 0 & 1 \end{bmatrix} ,$$

and build a 5-rowed matrix $A := A_I(2) \times A_T(2) \times A_T(1)$. If $F \prec A$, then $F \prec I \times T \times T$. If we do this for every possible partition of 5, we get the desired result.

To find $X(F)$ we can build a tree of possibilities of products and prune whenever we hit a node that already has configuration F:


$\emptyset$
$\mathcal{I}$
$\mathcal{I}^c$
$\mathcal{T}$
$\mathcal{I} \times \mathcal{I}$
$\mathcal{I} \times \mathcal{I}^c$
$\mathcal{I} \times \mathcal{T}$
$\mathcal{I}^c \times \mathcal{I}^c$
$\mathcal{I}^c \times \mathcal{T}$
$\mathcal{T} \times \mathcal{T}$
$\mathcal{I} \times \mathcal{I} \times \mathcal{I}$
...
. . .

We return the product with the largest number of factors for which F is not a subconfiguration of a product of this form, observing that if F is a subconfiguration of a product $P_r(a,b,c)$, then it will be a subconfiguration in any product $P_r(a',b',c')$ with $a' \geq a$, $b' \geq b$ and $c' \geq c$.

4.4 Finding Boundary Cases and Classifying Configurations

Given number of rows s and a number k, we wish to find all *boundary* cases, that is maximal and minimal F such that $X(F) = k$. We make use of the program described in the previous section to find $X(F)$ for many different F. The boundary cases for small s and k can be found in [Ans] and [Rag11].

The method we use is very straightforward: start adding columns, one by one. Build a tree of configurations, where the children of a configuration F are the ones that consist of F plus a column. Find $X(F)$ for each configuration in the tree. Store all those configurations F for which $X(F) = k$, and then find only the maximal and minimal configurations in the ordering $\prec$. If at some point we add a column and the bound jumps to $k + 1$ or higher, discard and go to the next configuration. Because of Proposition 2.3, we only need to consider cases where columns are repeated at most twice.

Unfortunately this method is very slow because the same configuration is searched multiple times, since each configuration may have many representatives. To get rid of repetition we might check for equivalence of configurations against everything we have stored so far. But checking if two configurations are equivalent is usually slow, so doing it every time is also very slow.

What seems to work best is to do check for equivalence, but only up to a point. For example, we can consider all pairs of columns, test those and only take one representative of each equivalence class. Then from each pair, start building the tree as described above. After that, it will be relatively unlikely that two configurations we search are equivalent, so the amount of repetition will be relatively low. Of course, much repetition will still occur, but much less than in the original tree.

Notice that it isn't as critical that classifying configurations be a fast calculation. We need to do it once for every s and k, but no more. Once we know the maximal and minimal quadratics for five rows, we never need to calculate them again. Other calculations, such as finding $X(F)$ or What Is Missing, have to be performed for each configuration (or family) we wish to study, so decreasing the running time is more of a priority in those cases since faster programs allow us to study bigger configurations.